UNITED STATES PATENT APPLICATION

For

ACCESSING FIRMWARE OF A REMOTE COMPUTER SYSTEM USING A REMOTE FIRMWARE INTERFACE

Inventors:

Michael A. Rothman
Vincent J. Zimmer
Mark S. Doran

Prepared by:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard
Los Angeles, CA 90025-1026
(206) 292-8600

Attorney's Docket No.: 42P16431

"Express Mail" mailing label number: EV320119104US
Date of Deposit: June 26, 2003
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450

Dominique Valentino
(Typed or printed name of person mailing paper or fee)

Dominique Valentino
(Signature of person mailing paper or fee)

6-26-03
(DATE SIGNED)

ACCESSING FIRMWARE OF A REMOTE COMPUTER SYSTEM USING A REMOTE FIRMWARE INTERFACE

FIELD OF THE INVENTION

[0001] The field of invention relates generally to computer systems and, more specifically but not exclusively, relates to accessing the firmware of a remote computer system using a remote firmware interface (RFI).

BACKGROUND INFORMATION

[0002] During a computer system startup, the computer system is self-tested and initialized through loading and execution of system firmware. Under personal computer (PC) architectures, this firmware is commonly referred to as the system's Basic Input/Output System (BIOS). In a typical PC architecture, the BIOS is the firmware that runs between the processor reset and the first instruction of the Operating System (OS) loader. The BIOS also acts as an interface between software and hardware components of a computer system during the OS runtime. As computer systems have become more sophisticated, the operational environment between the application and OS levels and the hardware level is generally referred to as the firmware or the firmware environment.

[0003] Today's systems allow for remote access to computers over a network. For example, Wired for Management (WfM) is a specification from the Intel Corporation that describes the performance of certain computer configuration and maintenance functions over a network. Using WfM, the installed hardware and

software of a remote computer can be determined and monitored in real time by another computer, such as a server. A feature called remote wakeup (RWU) minimizes unnecessary use of system bandwidth by keeping unused machines online only when they are needed according to a pre-planned schedule. Access to distant computers can be monitored and regulated. A server can access a remote computer to perform tasks such as repair corrupted files and programs, update the operating system, update virus programs, back up files and monitor the remote computer's health.

[0004] While the concept of remote invocation has been implemented in operating system environments, access to the firmware of a remote computer regardless of the presence of an operating system currently has limited capabilities. Without the ability to have flexible entry into the firmware infrastructure of the remote computer, a caller computer (e.g., a server) is dependent upon the data that the firmware of the remote computer is programmed to provide the caller. In some networks, the remote's firmware is limited to network boot operations and echoing of the remote system screen data at the caller's monitor. Other remote systems have specific boot-agents that communicate back to their caller in some proprietary fashion. However, these systems are vendor-specific and do not allow a single program to control remote systems in a similar manner when the remote systems come from a variety of vendors.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The present invention is illustrated by way of example and not limitation in the accompanying figures.

[0006] Figure 1 is a flowchart illustrating the logic and operations performed by a computer system under the Extensible Firmware Interface (EFI) framework standard in accordance with one embodiment of the present invention.

[0007] Figure 2 is a schematic diagram of a caller computer and a remote computer in accordance with one embodiment of the present invention.

[0008] Figure 3 is a flowchart illustrating the logic and operations performed by one embodiment of the invention to access firmware of a remote computer.

[0009] Figure 4 is a schematic diagram illustrating a computer system that may be used to practice an embodiment of the present invention.

DETAILED DESCRIPTION

[0010] Embodiments of a method to access firmware of a remote computer and computer apparatus for implementing the method are described herein. In the following description, numerous specific details are set forth, such as embodiments pertaining to the Extensible Firmware Interface (EFI) framework standard, to provide a thorough understanding of embodiments of the invention. One skilled in the relevant art will recognize, however, that the invention can be practiced without one or more of the specific details, or with other methods, components, materials, etc. In other instances, well-known structures, materials, or operations are not shown or described in detail to avoid obscuring aspects of the invention.

[0011] Reference throughout this specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, the appearances of the phrases "in one embodiment" or "in an embodiment" in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

[0012] In accordance with aspects of the invention, a mechanism to access the firmware of a remote computer system using a Remote Firmware Interface (RFI) is disclosed. With RFI, one can exercise program code in the firmware environment of a remote computer under the direct control of another computer. A caller computer, such as a server, has the ability to access data and initiate firmware procedures of a

remote machine, such as a network client. Requested tasks are performed under the control of firmware of the remote computer and independent of the remote computer's operating system. Thus, through RFI, the remote computer can process tasks assigned by a caller during pre-boot, OS runtime, or even after the operating system has crashed.

[0013] In accordance with one embodiment, the remote firmware interface may be implemented under an extensible firmware framework standard known as the Extensible Firmware Interface (EFI) (specifications and examples of which may be found at http://developer.intel.com/technology/efi). EFI is a public industry specification that describes an abstract programmatic interface between platform firmware and shrink-wrap operating systems or other custom application environments. The EFI standard include provisions for extending BIOS functionality beyond that provided by the BIOS code stored in a platform's BIOS device (e.g., flash memory). More particularly, EFI enables firmware, in the form of firmware modules and drivers, to be loaded from a variety of different resources, including primary and secondary flash devices, option ROMs (Read-Only Memory), various persistent storage devices (e.g., hard disks, CD-ROMs (Compact Disk-Read Only Memory), etc.), and even over computer networks.

[0014] Figure 1 shows an event sequence diagram 100 used to illustrate operations performed by a computer system under one implementation of the EFI framework standard. In a block 102, power is added to a computer system and the platform initialization phase begins. This phase provides a standardized method of loading and invoking specific initial configuration routines for the processor, chipset,

and motherboard. The initialization phase is responsible for initializing enough of the system to provide a stable base for the follow on phases. Initialization of the platforms core components, including the processor, chipset and main board (i.e., motherboard) is performed during this phase. This phase is also referred to as the "early initialization" phase. Typical operations performed during this phase include the POST (power-on self test) routine, and discovery of platform resources, such as memory.

[0015] Once the EFI firmware is initialized, it passes control to a Boot Manager, as depicted in a block 104. The Boot Manager is a firmware policy engine that loads EFI applications and EFI drivers into memory in a pre-defined order. EFI applications are modular code that may perform specific tasks outside of the OS environment. EFI drivers are modules of code that provide device support during the boot process or during OS runtime.

[0016] The Boot Manager also produces an EFI System Table. The EFI System Table contains standard input and output handles for an EFI application, as well as pointers to the Boot Services and Runtime Services Tables. The EFI System Table may also contain pointers to other standard tables such as the Advanced Configuration and Power Interface (ACPI) Table.

[0017] The EFI specification defines two types of EFI Services: Boot Services and Runtime Services. Boot Services are not available after an OS Loader has taken control of the platform. Boot Services include, but are not limited to, Memory Services, Event and Time Services, and Image Services. In contrast to Boot Services, Runtime Services are available both during pre-boot and OS runtime

operations. Runtime Services include, but are not limited to, Reset Services, Status Code Services, and Virtual Memory Services.

[0018] Functions defined in an EFI-compliant system are called through pointers in common, architecturally defined, calling conventions found in C compilers. Pointers to the various global EFI functions are found in the Boot Services table and the Runtime Services table that are located via the EFI System Table. Pointers to other functions are located dynamically through device handles. A device handle points to a list of one or more protocols that can respond to requests for services for a given device referred to by the handle. A protocol typically contains a set of protocol interfaces that are published to the computer system. System components can access a protocol by calling the protocol interface. A protocol interface also provides for passing an argument (a.k.a., a parameter) to and receiving an argument from the protocol. In one embodiment, a protocol interface is referred to as an Application Program Interface (API).

[0019] In a block 106, the EFI OS Loader is loaded into memory. The OS Loader is the first piece of operating system code loaded by the firmware to initiate the OS boot process. The OS Loader is loaded at a fixed address and then executed. In one embodiment, a user is presented with a user interface generated by firmware to select the operating system to be booted. An EFI OS Loader can support multiple OS boot options. An EFI OS Loader can also support legacy boot options such as booting from the A: drive or C: drive of the computer system.

[0020] Once the OS Loader is loaded, control of the computer system is transferred to the OS Loader, as shown in a block 108. If the OS Loader

successfully loads its operating system, it can take control of the platform by using the Boot Service *ExitBootServices*. After successfully calling *ExitBootServices*, all boot services in the system are terminated. From this point the OS Loader is responsible for continued operation of the computer system. If the load of the OS fails, then the OS Loader returns an *Exit* call and control of the system is returned to the EFI firmware.

[0021] Figure 2 illustrates an embodiment of a network 200 according to one embodiment of the present invention. Such a network may include, but is not limited to, a Local Area Network (LAN), a Wide Area Network (WAN), or a collection of LANs and WANs, such as an enterprise intranet or the Internet. The network connection may comprise a wired connection (e.g., Ethernet, token ring, etc.), a wireless connection including optical systems, satellite transmissions, or the like, or any combination thereof.

[0022] Network 200 includes a caller computer 202 and a remote computer 204. Generally, caller computer 202 and remote computer 204 include, but are not limited to, a personal computer, a network server, a network workstation, a portable computer, a handheld or palmtop computer, a personal digital assistant (PDA), or the like. In one embodiment, caller computer 202 and remote computer 204 are configured in a similar manner to an exemplary computer system discussed below in conjunction with Figure 4. In one embodiment, the caller computer 202 is a server and remote computer 204 is a client in network 200. The network 200 is shown with one caller and one remote for clarity, but it will be understood that network 200 could

contain more computer systems, each computer system capable of acting as a caller or a remote computer.

[0023] Caller computer 202 sends a request packet 206 onto the network 200 addressed for remote computer 204. In one embodiment, the request packet 206 is sent in real-time. In another embodiment, the request packet 206 is a scheduled event that occurs automatically according to a pre-planned schedule. After remote computer 204 receives the request packet 206, the remote computer 204 performs the action as defined in the request packet 206. After completing the requested action, the remote computer 204 prepares a response packet 208. The response packet 208 may contain data requested by the caller computer 202 or simply confirmation that the requested task was successfully completed or failed due to some error.

[0024] Figure 2 also shows a listening mechanism 210 that is part of remote computer 204 and is the manner by which the remote computer 204 receives request packets. In one embodiment, the remote computer 204 uses a polling mechanism to listen for request packets; while in another embodiment, the remote computer 204 uses an interrupt mechanism. In a polling implementation, the remote computer periodically checks a particular place to see if it has received a request packet. In one embodiment, the remote determines if a Network Interface Card (NIC) has received and stored a request packet. In another embodiment, the remote checks a Universal Asynchronous Receiver/Transmitter (UART) to determine if it has received and stored a request packet from its serial port connection. If the remote determines it has received a packet, then the remote's firmware processes

the request accordingly and sends a response packet back to the caller. If the remote has not received a request packet, then the remote's firmware continues performing other duties as necessary. The firmware will continually check for a request packet after a pre-determined time period.

[0025] In an interrupt implementation, the receipt of a request packet at the remote computer 204 creates an interrupt. The remote's firmware stops its current task to handle the request packet and send a response packet back to the caller. In one embodiment, the request packet signals a System Management Interrupt (SMI) for the remote to enter a System Management Mode (SMM).

[0026] SMM is a special mode for handling system wide functions and is intended for use only be system firmware, and not by an OS or an application. When SMM is invoked through an SMI, the processor saves the current state of the processor and switches to a separate operating environment contained in system management random access memory (SMRAM). While in SMM, the processor executes SMI handler code to perform operations. When the SMI handler has completed its operations, it executes a resume instruction. This instruction causes the processor to reload the saved state of the processor, switch back to protected or real mode, and resume executing the interrupted application or OS tasks.

[0027] It will be understood that the request packet 206 can be received and implemented by the remote computer 204 independent of the state of the OS because the request packet is read and executed under the control of the remote's firmware. For example, a system administrator wishes to remotely debug a program that is installed on a plurality of remote computers on a network. The system

administrator writes a script and uses a Telnet application to contact a group of remote computers and request each remote to report the contents of a particular memory address.

[0028] In another example, a system administrator pushes a firmware update to a group of remotes on a network. The firmware update execution file is included in the request packet sent to each remote. After receiving the request packet, each remote computer loads and executes the firmware update. The response packet from each remote then informs the system administrator whether the firmware update installed successfully or encountered a problem.

[0029] In another example, a system administrator is alerted that the OS running on a remote computer has become hung. In response, the system administrator sends a request packet from the server to the hung remote. The request packet asks the remote's firmware to perform a core dump and send the results back to the server. In one embodiment, the core dump captures the settings of the remote such as the memory contents, the processor settings, device settings, and the like. Even though the OS has crashed, the request packet can still be answered by the remote because the request packet is processed in the firmware environment. After the core dump is sent to the server in a response packet, the system administrator resets the remote through another RFI packet. Thus, the system administrator can return the remote computer to service and still be able to analyze the core dump at a more convenient time.

[0030] A flowchart 300 illustrating further details of logic and operations performed in accordance with one embodiment of the present invention is shown in

Figure 3. The flowchart 300 begins in a block 302, which corresponds to a system startup event, i.e., a cold boot or a system reset of a remote computer.

[0031] In response to the startup event, pre-boot initialization of the remote computer will begin through loading and execution of system boot instructions stored in the computer system BIOS firmware. In one embodiment, the system boot instructions will begin initializing the platform by conducting a Power-On Self-Test (POST) routine, initializing system board functions, checking for any expansion boards that hold additional BIOS code, and loading such BIOS code if any is found.

[0032] During the system startup, a network interface of the remote is initialized, as shown in a block 304. At this point, the remote computer is able to communicate with a network and is eligible to receive request packets from a caller computer. Continuing to a block 306, the mechanism to listen for request packets is initialized. In one embodiment, the listening mechanism is a polling scheme, while in another embodiment, the listening mechanism is an interrupt scheme.

[0033] Referring again to Figure 3, once the listening mechanism has been initialized, the logic proceeds to a block 308 where a request packet is received from a caller by the remote. After receiving the request packet, the logic proceeds to a block 310, where the remote processes the request packet. The request packet contains one or more tasks for the remote to perform. As described below, a task includes instructing the firmware to execute code stored on the remote or another computer accessible to the remote. A task also includes providing the remote computer with code in the request packet for the remote to execute. In one embodiment, if the request packet contains a plurality of separate tasks for the

firmware to perform, then the firmware returns one response packet that includes a response for each of the plurality of assigned tasks.

[0034] It will be appreciated that RFI allows for interrogation of a remote machine without the necessity of constructing a pre-defined binary image for the task. In one embodiment, scripting is initiated from a caller computer. A scripting language is a high-level programming language that is interpreted instead of compiled before execution. This is different from pushing code to run on a remote computer because once the program is running on the remote, the caller usually cannot interrupt the process. With the RFI mechanism, the caller computer is not limited to pre-defined capabilities that are stored on the remote computer, such as in a PXE ROM (PreBoot Execution Environment Read-Only Memory). RFI enables flexibility in interacting with the remote system as if an operator was writing code line-by-line while the remote is running. This interaction is performed at the firmware level and independent of an operating system of the remote computer. In one embodiment of an EFI-compliant system, a scripting language is used to interrogate a remote computer to determine what protocols are available to the remote. Using this information, the remote is instructed to perform a task using a particular protocol. In another embodiment, after using the scripting language to determine what protocols are stored on the remote computer, the caller computer loads a driver onto the remote computer. The caller computer then tells the remote computer to install the protocol corresponding to the driver, such as by issuing an *InstallProtocolInterface* command.

[0035] In one embodiment, the remote's firmware analyzes the request packet header to determine what type of request packet was sent. In one embodiment in an EFI-compliant system, such a packet header is defined as follows:

```
typedef struc {
    EFI_GUID    PacketDefinition;    //what type of packet?
    UINTN       PacketLength;        //size of the entire packet
    //UINT8     PacketData[]         //packet data
    } PACKET_HEADER;
```

[0036] One type of request packet is a firmware interface packet. An interface packet is used to call a pre-defined function available to the firmware of the remote computer. In an EFI-compliant system embodiment, an interface packet includes a request to run a protocol interface function. An embodiment of an interface packet of an EFI-compliant system is defined as follows:

```
typedef struc {
    EFI_GUID    InterfaceType;       //which protocol?
    //UINT8     Parameters[];        //parameter stack
} INTERFACE_PACKET;
```

[0037] Another type of request packet is a memory packet to access a memory address of the remote computer. One embodiment of an interface packet is defined as follows:

```
typedef struc {
    UINTN       Address;     //what memory address?
    BOOLEAN     Write;       //TRUE = write memory
                             //FALSE = read memory
    UINTN       BufferSize;  //size of the packet buffer
    //UINT8     Buffer[];    //buffer to read/write
} MEMORY_PACKET;
```

[0038] Another type of request packet is a data structure packet to access data contained in a data structure accessible by the firmware of the remote computer. In

an embodiment of an EFI-compliant system, the data structure packet is used to access the data maintained in an EFI table. In one embodiment, a table packet is defined as follows:

```
typedef struc {
    TABLE_TYPE      WhichTable;     //enum of tables
    CHILD_TYPE      TableEntry;     //enum of table entries
    SUBCHILD_TYPE   TableEntry;     //enum of table entries
    //UINT8         Parameters[];   //parameter stack
} TABLE_PACKET;
```

[0039] Referring to flowchart 300, after the remote processes the request packet in block 310, the task requested by the request packet is executed by the remote's firmware, as illustrated in a block 312. After completion of the task, the remote prepares and returns a response packet to the caller computer, as depicted in a block 314. In one embodiment, if the remote is unable to complete a task requested by a caller, the remote will return a request packet containing an error message for the caller. In one embodiment, such an error message includes a notice that the assigned task has failed and information as to the source of the error.

[0040] Figure 4 illustrates an embodiment of an exemplary computer system 400 to practice an embodiment of the invention described above. Computer system 400 is generally illustrative of various types of computer devices, including personal computers, laptop computers, workstations, servers, etc. For simplicity, only the basic components of the computer system are discussed herein. Computer system 400 includes a chassis 402 in which various components are housed, including a floppy disk drive 404, a hard disk 406, a power supply (not shown), and a motherboard 408. The motherboard 408 includes a memory 410 coupled to one or more processors 412. Memory 410 may include, but is not limited to, Dynamic

Random Access Memory (DRAM), Static Random Access Memory (SRAM), Synchronized Dynamic Random Access Memory (SDRAM), Rambus Dynamic Random Access Memory (RDRAM), or the like. Processor 412 may be a conventional microprocessor including, but not limited to, an Intel Corporation x86, Pentium, or Itanium family microprocessor, a Motorola family microprocessor, or the like. Hard disk 406 may comprise a single unit, or multiple units, and may optionally reside outside of computer system 400.

[0041] The computer system 400 also includes a non-volatile memory device on which firmware is stored. Such non-volatile memory devices include a ROM device 420 or a flash device 422. Other non-volatile memory devices include, but are not limited to, an Erasable Programmable Read Only Memory (EPROM), an Electronically Erasable Programmable Read Only Memory (EEPROM), or the like. The computer system 400 may include other firmware devices as well (not shown).

[0042] A monitor 414 is included for displaying graphics and text generated by firmware, software programs and program modules that are run by computer system 400, such as system information presented during system boot. A mouse 416 (or other pointing device) may be connected to a serial port, USB (Universal Serial Bus) port, or other like bus port communicatively coupled to processor 412. A keyboard 418 is communicatively coupled to motherboard 408 in a similar manner as mouse 416 for user entry of text and commands. In one embodiment, computer system 400 also includes a network interface card (NIC) or built-in NIC interface (not shown) for connecting computer system 400 to a computer network 430, such as a local area network (LAN), wide area network (WAN), or the Internet. In one embodiment

network 430 is further coupled to a remote computer 435, such that computer system 400 and remote computer 435 can communicate. In one embodiment, a portion of the computer system's firmware is loaded during system boot from remote computer 435.

[0043] The illustrated embodiment further includes an optional add-in card 424 that is coupled to an expansion slot of motherboard 408. In one embodiment, add-in card 424 includes an Option ROM 426 on which firmware is stored. Computer system 400 may also optionally include a compact disk-read only memory ("CD-ROM") drive 428 into which a CD-ROM disk may be inserted so that executable files, such as an operating system, and data on the disk can be read or transferred into memory 410 and/or hard disk 406. Other mass memory storage devices may be included in computer system 400.

[0044] In another embodiment, computer system 400 is a handheld or palmtop computer, which are sometimes referred to as Personal Digital Assistants (PDAs). Handheld computers may not include a hard disk or other mass storage, and the executable programs are loaded from a corded or wireless network connection into memory 410 for execution by processor 412. A typical computer system 400 will usually include at least a processor 412, memory 410, and a bus (not shown) coupling the memory 410 to the processor 412.

[0045] It will be appreciated that in one embodiment, computer system 400 is controlled by operating system software that includes a file management system, such as a disk operating system, which is part of the operating system software. For example, one embodiment of the present invention utilizes Microsoft Windows® as

the operating system for computer system 400. In another embodiment, other operating systems such as, but not limited to, the Apple Macintosh operating system, the Linux operating system, the Microsoft Windows CE® operating system, the Unix operating system, the 3Com Palm operating system, or the like may also be use in accordance with the teachings of the present invention.

[0046] Thus, embodiments of this invention may be used as or to support a firmware and software code executed upon some form of processing core (such as processor 412) or otherwise implemented or realized upon or within a machine-readable medium. A machine-readable medium includes any mechanism that provides (i.e., stores and/or transmits) information in a form readable by a machine (e.g., a computer, network device, personal digital assistant, manufacturing tool, any device with a set of one or more processors, etc.). For example, a machine-readable medium includes, but is not limited to, recordable/non-recordable media (e.g., a read only memory (ROM), a random access memory (RAM), a magnetic disk storage media, an optical storage media, a flash memory device, etc.). In addition, a machine-readable medium can include propagated signals such as electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.).

[0047] The above description of illustrated embodiments of the invention, including what is described in the Abstract, is not intended to be exhaustive or to limit the invention to the precise forms disclosed. While specific embodiments of, and examples for, the invention are described herein for illustrative purposes,

various equivalent modifications are possible within the scope of the invention, as those skilled in the relevant art will recognize.

[0048] These modifications can be made to the invention in light of the above detailed description. The terms used in the following claims should not be construed to limit the invention to the specific embodiments disclosed in the specification and the claims. Rather, the scope of the invention is to be determined entirely by the following claims, which are to be construed in accordance with established doctrines of claim interpretation.